

Architecture of the AMD Quad Core CPUs

Brian Waldecker, Ph.D. | April 13, 2009
Senior Member of Technical Staff
Performance CoE
AMD, Austin



Outline

1. AMD and Cray Roadmaps
2. Opteron™ Multi-Core Architectural Overview
3. NUMA: Multi-socket and Multi-core considerations
4. Programming Hints and Performance Case Studies



Cray and AMD Tools for Breakthrough Science

Many Sites, Multiple Disciplines

ARL

- 10,400 core XT5 (Barcelona)
- 1,952 cores XT5 (Barcelona)

UT et. al. NSF Track II – (big!)

NERSC Franklin – 356 TF

ARSC – 31.8 TF XT5

Sandia Red Storm – 284 TF XT4

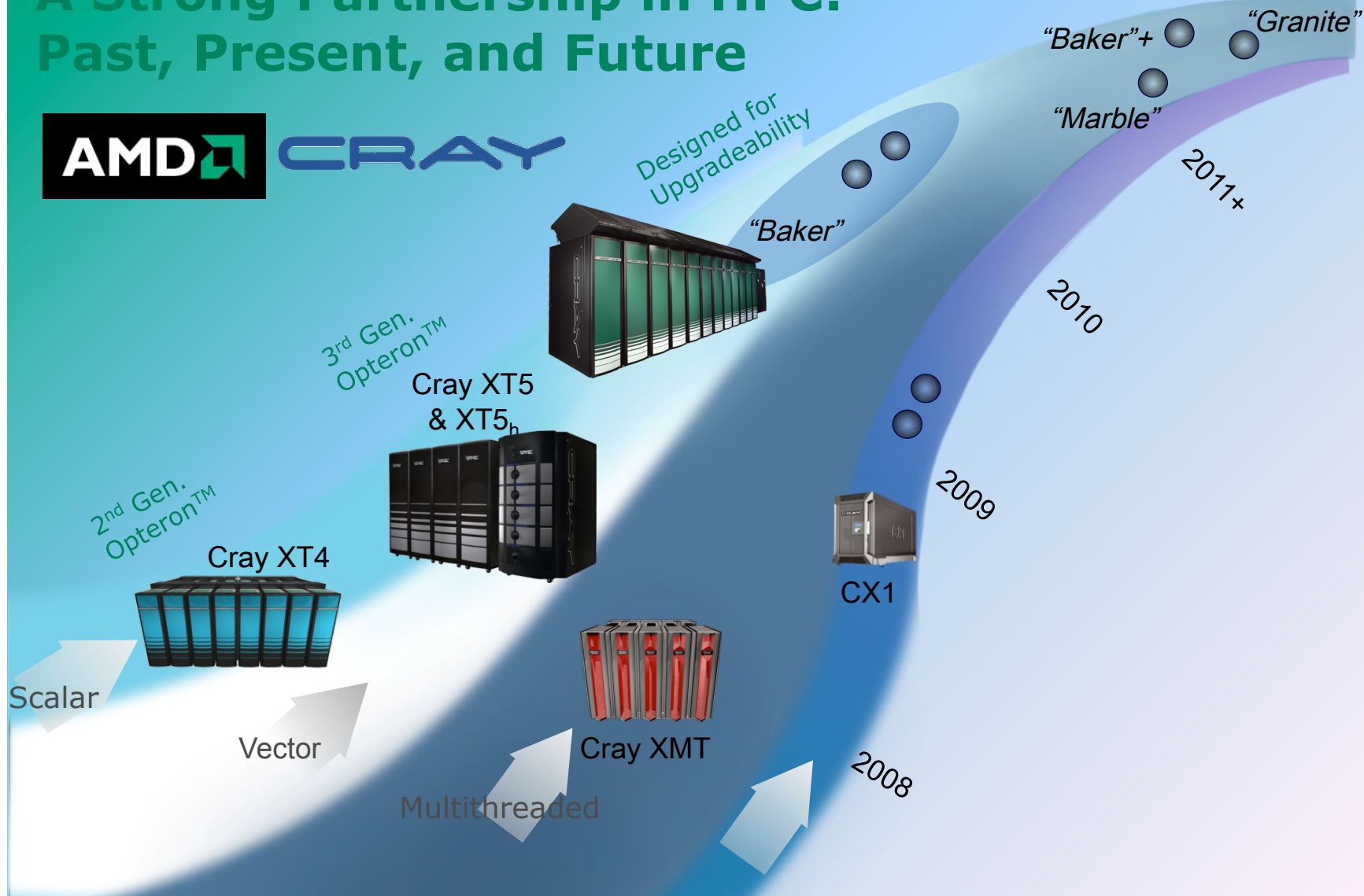
NAVO – 117 TF XT5

University of Bergen - 50 TF XT4

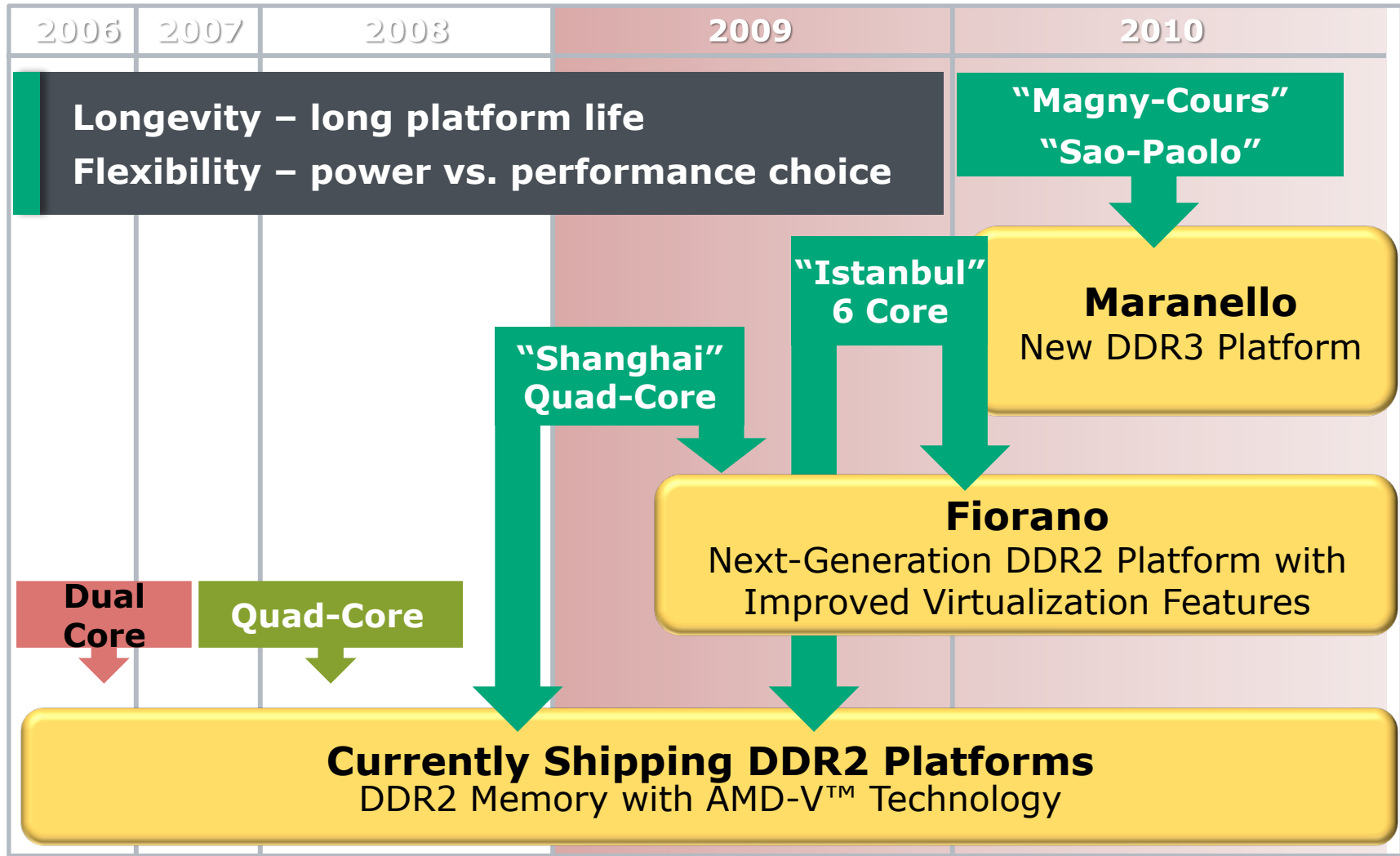
CSC Finland – 86.7 TF enroute to 100+ TF



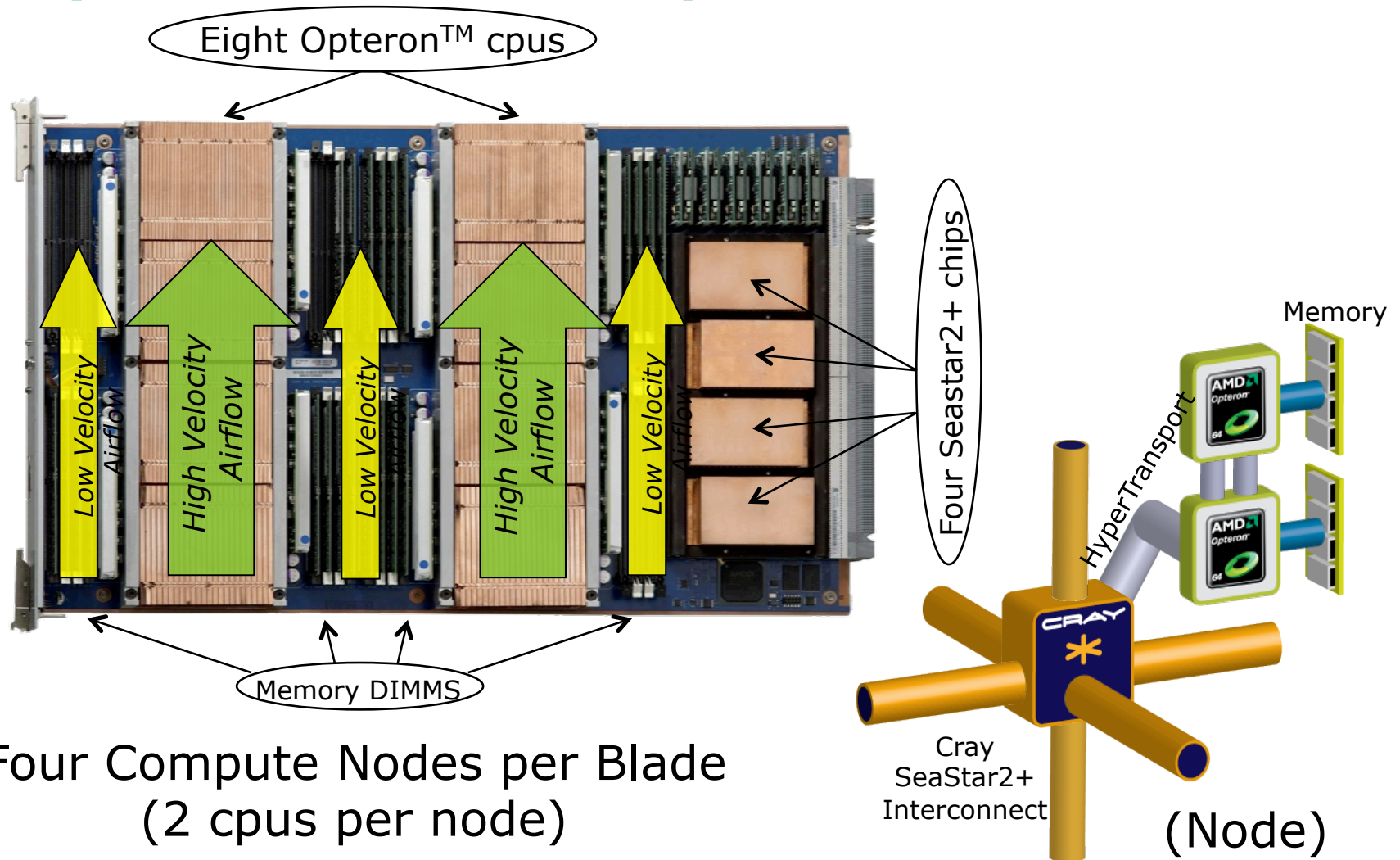
A Strong Partnership in HPC: Past, Present, and Future



AMD Cross-Generation x86 Server Platforms Roadmap



Cray XT5 Blade and Compute Node



“Barcelona” to “Shanghai”

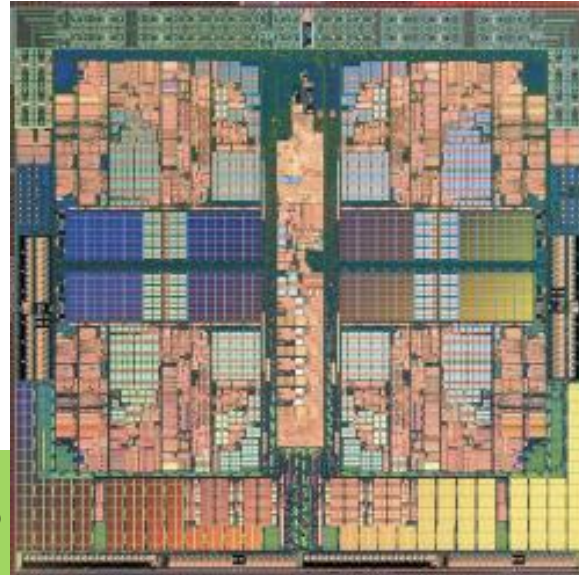
What's New

- 65nm to 45 nm
 - Higher GHz
 - Lower Power
- 2M L3 to 6M L3
- Lower Latency L3
- Prefetcher tweaks
- DDR2-800 support
- HT3 HyperTransport™

Barcelona

```
vendor_id   : AuthenticAMD
cpu family  : 16
model       : ②
model name  : Quad-Core AMD Opteron(tm) Processor 8356
stepping    : ③
```

barcelona

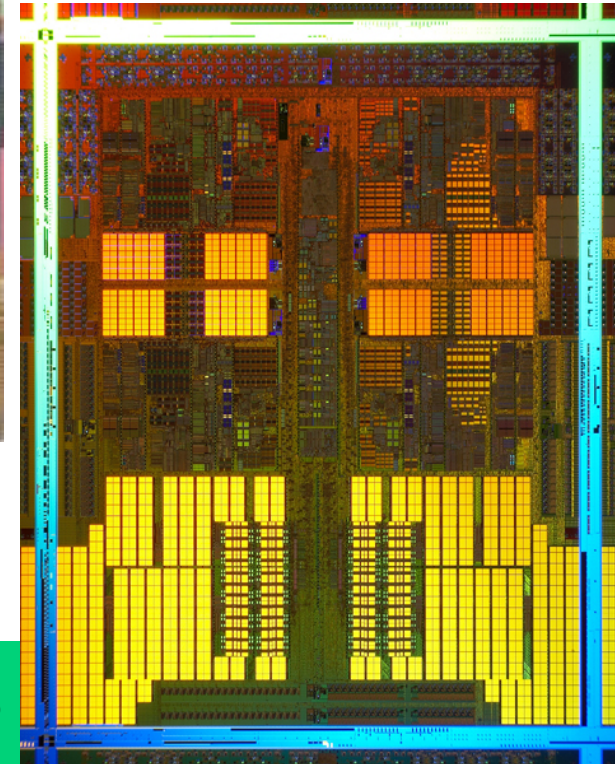


(note: not to same scale)

Shanghai

```
vendor_id   : AuthenticAMD
cpu family  : 16
model       : ④
model name  : Quad-Core AMD Opteron(tm) Processor 8387
stepping    : ②
```

shanghai



"Barcelona" to "Shanghai"

"Barcelona" Features

65nm Technology

AMD Balanced Smart Cache

HyperTransport™ Technology 1.0 @
up to 8GB/s*

AMD Memory Optimizer
Technology

Drop-in Upgradeability
Investment Protection

Cache ECC & scrubbing, CPU and
Northbridge watchdogs

Products range from 2.0 to 2.3 GHz
(standard 75W power)

New With "Shanghai"

45nm Technology
Significantly reduced power &
increased frequency

L3 grows to 6MB (8MB total cache)
2x more expected to improve application
performance by 5-10%

HyperTransport™ Technology 3.0 @
up to 4.4GTs or 17.6GB/s*

DDR2-800 Memory Support
(Up to 10% greater delivered
memory bandwidth)

Continued Drop-in Upgradeability
Investment Protection

L3 cache Index Disable
(Designed to protect data against
L3 cache errors)

Products range from 2.3 to 2.7 GHz
(standard 75W power)

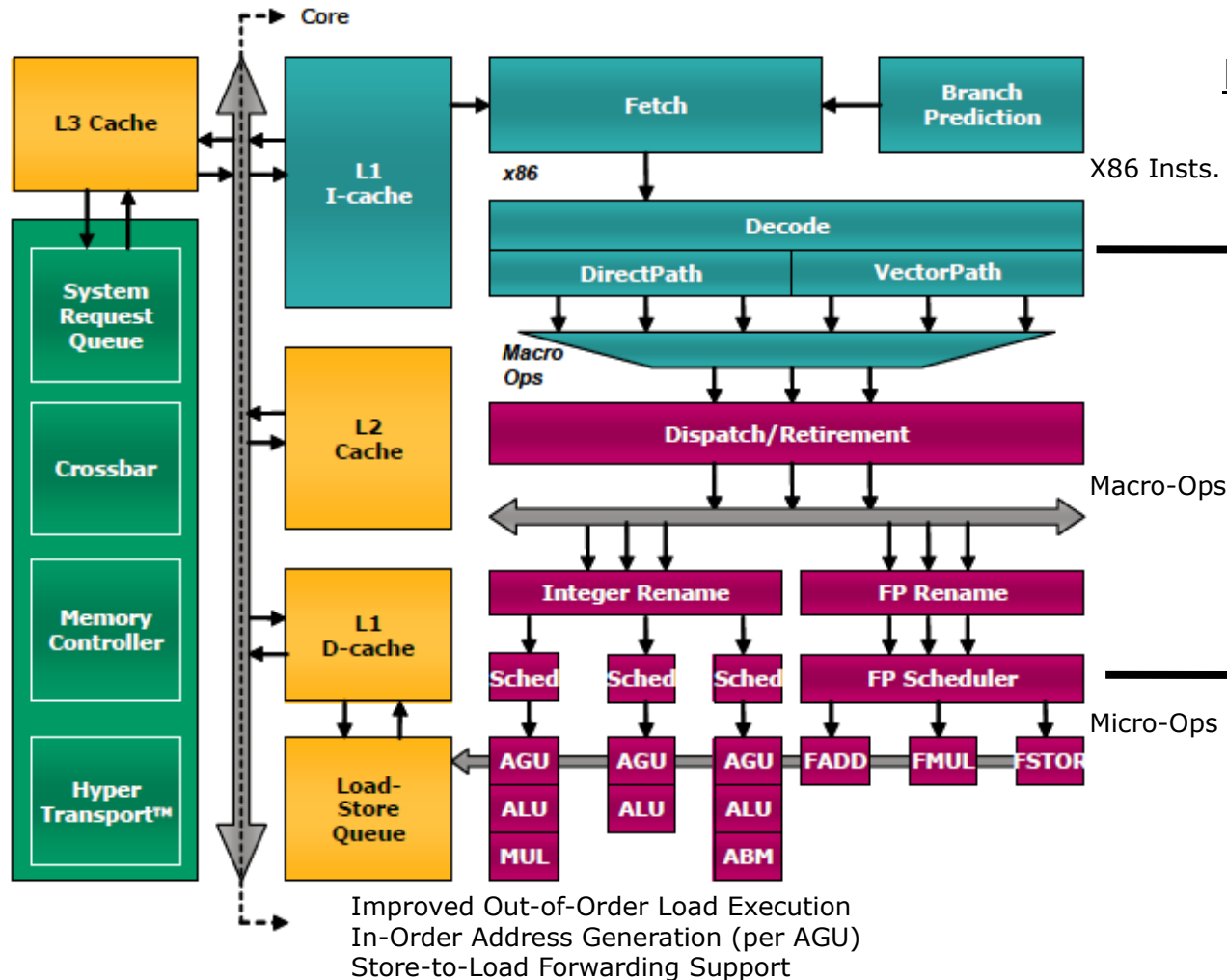
* bi-directional



Core Micro Architecture

FastPath? Macro-Ops? Micro-Ops?

Reference : Software Optimization Guide for
AMD Family 10h Processors,
Pub. #40546, Rev. 3.10 Feb 2009



Notes / Considerations

X86 Insts. Avoid having more than 2 or 3 branches per 16B of instructions.

Three Decode Categories (FastPath also called DirectPath)

- DirectPath Single - *best*
- DirectPath Double - *better*
- VectorPath (microcode) - *good*

Macro-Ops tracked ReOrder Buffer (ROB).

- 72 entries (3 wide x 24 deep)
- In-order dispatch, retirement

Micro-Ops issue from Sched to Execution Units

- "Sched" aka "Reserv. Station"
- Out-of-order issue
- FP scheduler shared across units
- INT Schedulers are "per unit"



Relating Micro-Architecture to Programming

Micro-architectural feature	Rely on Compiler Magic?	Coerce Compiler using flags	Tweak Src Code to help Compiler	Examples
Direct / uCode	Yes	Difficult	Difficult	<ul style="list-style-type: none"> • (not much programmer can do to control)
3 wide super-scalar design	Yes	Difficult	Sometimes Useful	<ul style="list-style-type: none"> • Computational Intensity of loops (CrayPAT). • Write Vectorizable loops. • Independent Ops. Vs Dependency Chains within code blocks.
Cache Sizes and Geometries	Yes	Yes	Yes	<ul style="list-style-type: none"> • Cache Blocking of Loops. • Array padding. • Prefetch and Streaming Store compiler flags
Branch Pred., Address Gen.	Yes	Difficult	Yes	<ul style="list-style-type: none"> • Unrolling & good branch-to-code density. • Help Compiler to Inline ("static" funcs. in C). • Hoist common code and order "if" statements for most common cases. • Simple addr. calcs before complicated ones.
Ld/St BW + # of Func. Units	Yes	Yes	Yes	<ul style="list-style-type: none"> • Computational Intensity of loops (CrayPAT) • Prefetch and Streaming Store compiler flags
Ld/St BW + Reg. File Size	Yes	Maybe		<ul style="list-style-type: none"> • Help with idiom recognition and use algorithmic knowledge. (e.g. grid sweeps) • Aliasing hints (via flags and careful ptr use)
Data Alignment	Yes	Yes	Yes	<ul style="list-style-type: none"> • Declares struct elements largest to smallest • Buffer padding and pointer adjustment.



TLB Review (Barcelona, Shanghai, Istanbul)

- Support for 1GB pagesize (4k, 2M, 1G)
- 48 bit physical addresses = 256TB (increase from 40bits on K8)
- Data TLB
 - L1 Data TLB
 - 48 entries, fully associative
 - all 48 entries support any pagesize
 - L2 TLB
 - 512 4k entries, and
 - 128 2M entries
- Instruction TLB
 - L1 Instruction TLB
 - fully associative
 - support for 4k or 2M pagesizes
 - L2 Instruction TLB



Data Prefetch: Review of Options

- **Hardware prefetching**

- DRAM prefetcher
 - tracks positive, negative, non-unit strides.
 - dedicated buffer (in NB) to hold prefetched data.
 - Aggressively use idle DRAM cycles.
- Core prefetchers
 - Does hardware prefetching into L1 Dcache.

- **Software prefetching instructions**

- MOV (prefetch via load / store)
- prefetcht0, prefetcht1, prefetcht2 (currently all treated the same)
- prefetchw = prefetch with intent to modify
- prefetchnta = prefetch non-temporal (favor for replacement)



Cache Hierarchy

Dedicated L1 cache

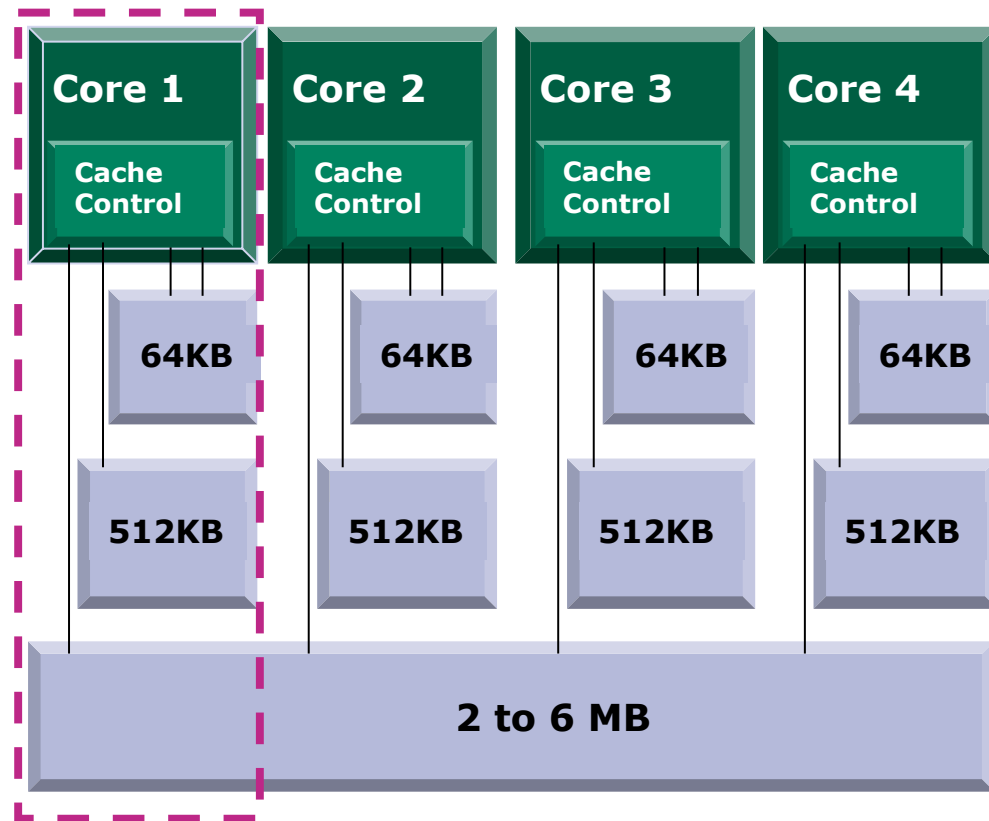
- 2 way associativity.
- 8 banks.
- 2 128-bit loads per cycle.

Dedicated L2 cache

- 16 way associativity.

Shared L3 cache

- 32 way (barcelona), 48 way (shanghai) associativity.
- fills from L3 leave likely shared lines in L3.
- sharing aware replacement policy.



Shanghai to Istanbul

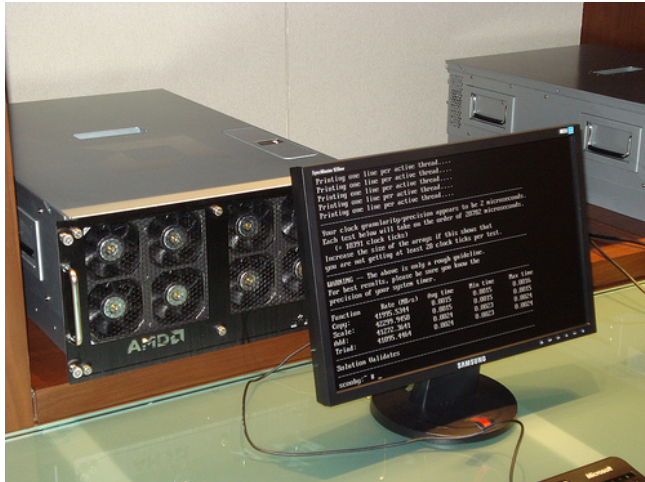
- 6 cores ($\sim 1.5X$ flops)
 - Same per core L1 & L2
 - Same shared L3
 - NB & Xbar upgrades (going from 4 to 6 cores)
- HT Assist – provides 3 probe scenarios
 - No probe needed
 - Directed probe
 - Broadcast probe
- Memory BW and latency improvement
 - Amount depends on platform and configuration
- Socket Compatibility



Socket Compatibility: 4P/16cores → 4P/24cores



HT Assist (Probe Filters)



...e per active thread....
...e per active thread....
...e per active thread....
...e per active thread....
...e per active thread....

clarity/precision appears to be 2 microseconds.
will take on the order of 20782 microseconds.
k ticks)

Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.

WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.

Function	Rate (MB/s)	Avg time	Min time	Max time
Copy:	41995.5344	0.0015	0.0015	0.0016
Scale:	42299.9458	0.0015	0.0015	0.0015
Add:	41272.3641	0.0024	0.0023	0.0024
Triad:	41095.4464	0.0024	0.0023	0.0024

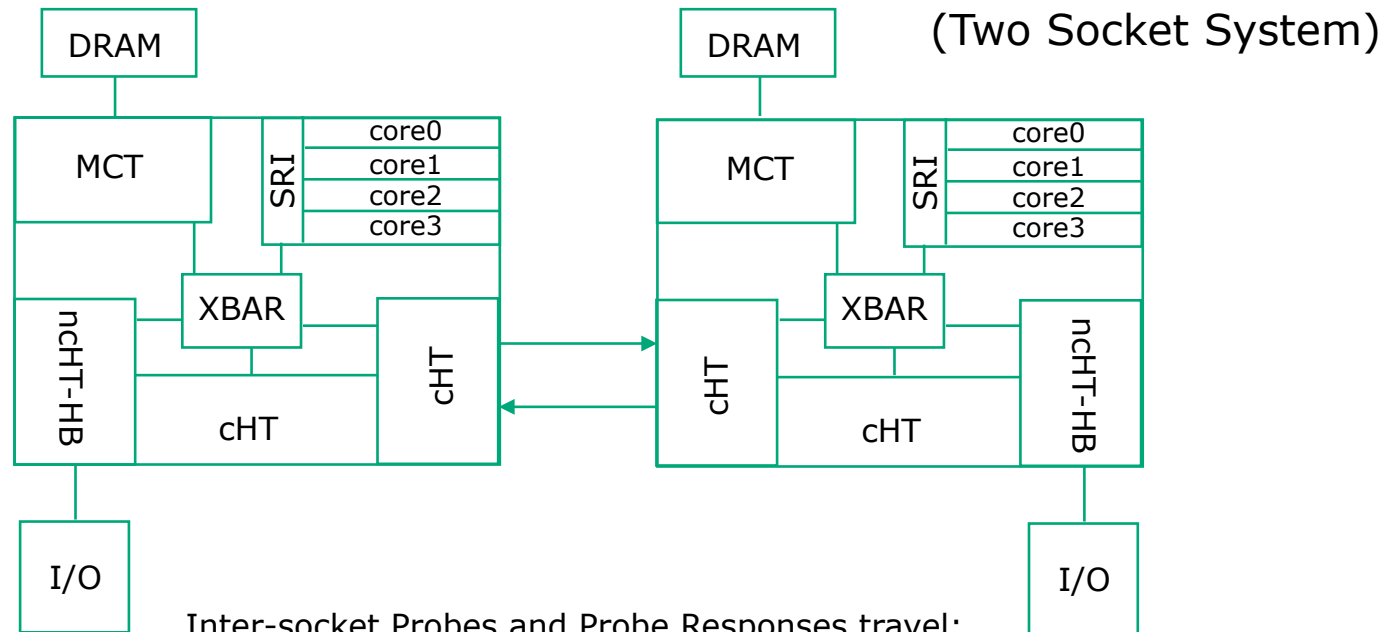
Solution Validates

scooby:~ #

SAMSUNG



Multi-Socket System Overview



Inter-socket Probes and Probe Responses travel:
SRI -> XBAR -> cHT -> cHT -> XBAR -> SRI

Probes Requests initiate at home memory node, but
return directly to node making initial memory request.

key:

cHT = coherent HyperTransport

nCHT = non-coherent HyperTransport

XBAR = crossbar switch

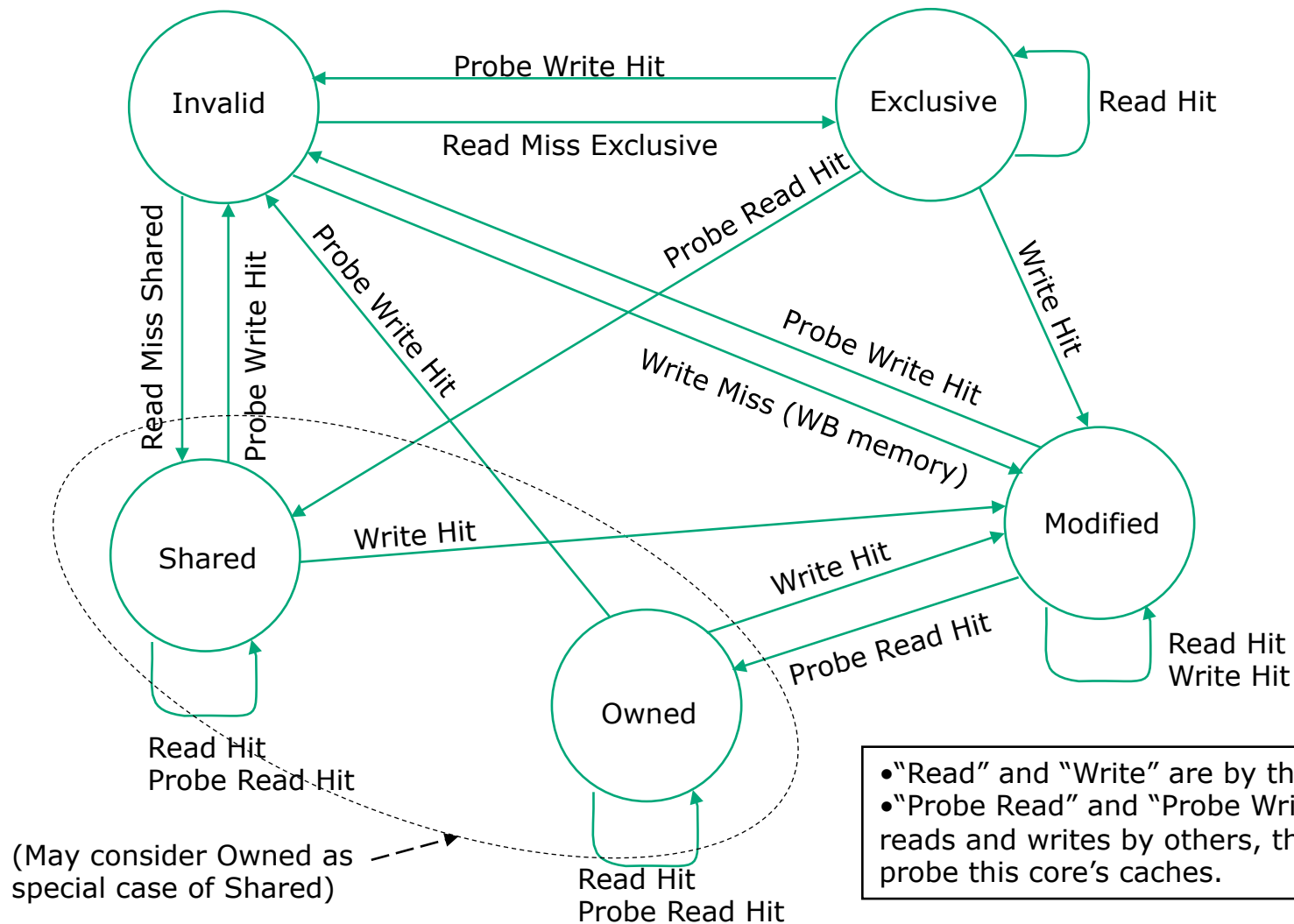
SRI = system request interface (memory access, cache probes, etc.)

MCT = memory controller

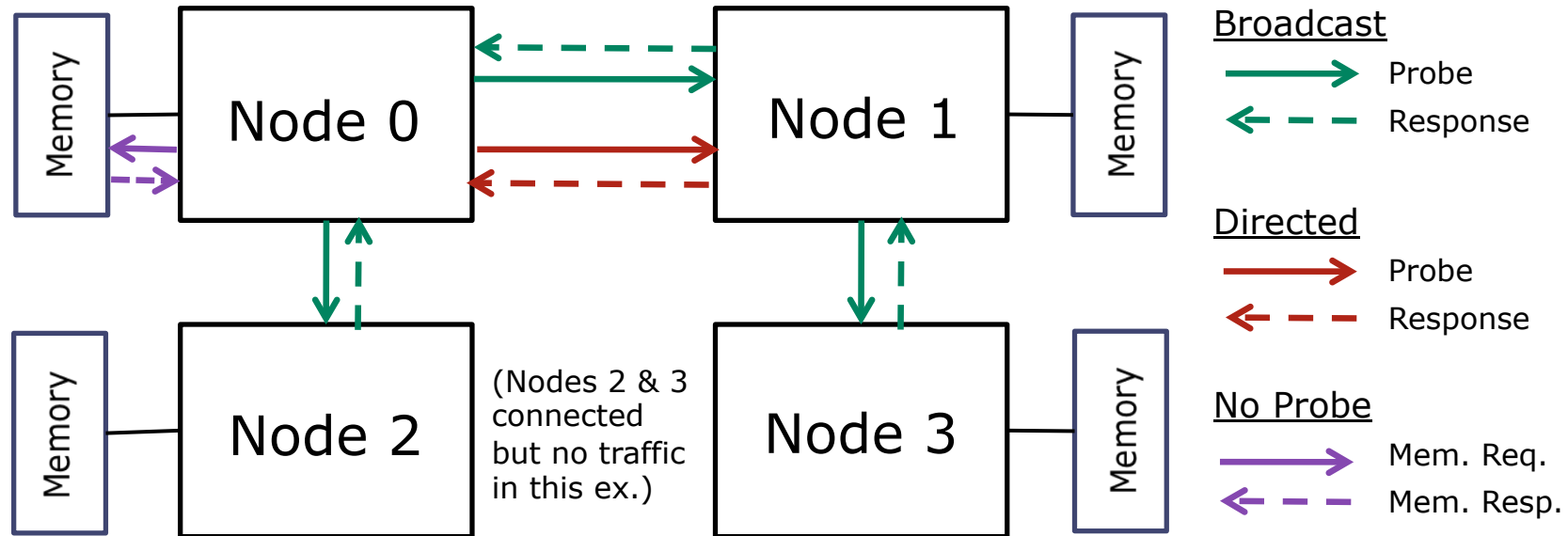
HB = host bridge (e.g. HT to PCI, SeaStar, etc.)



MOESI Cache Coherency Protocol



HT (Hyper Transport) Assist Benefits



- Tracks cacheline usage and eliminates much Probe Traffic from HT Fabric.
 - Cache misses going to memory often avoid probing entirely.
 - Write upgrade to M state often results in directed probes or no probes.
 - Read of shared data will often result in a directed probe.
 - Worst case requires broadcast (i.e. pre- HT Assist behavior).



Cache Coherency – Practical Advice

- Avoid shared read and shared write data in same cacheline.
- Avoid gratuitously modifying shared data.
 - Sharing aware L3 helps within a chip, but doesn't make such updates free.
 - Minimize false sharing where compiler has to play it safe.
- Requirement to wait for all probe responses means local memory and remote cache accesses have similar latencies.
 - Sometimes thinking of just memory is just fine.
 - Let library and compiler writers worry about being uber-clever.
- Aliasing, Aliasing, Aliasing of addresses. (Help the compiler).
 - If compiler's unsure about potential aliasing it must play it safe and generate extra stores and loads, instead of working only with registers.



Sharing Aware (Partially Inclusive) L3

- Inclusive vs. Exclusive Cache Paradigms
 - Inclusive: L3 contains L2 contains L1 (i.e. supersets).
 - Exclusive: L3, L2, and L1 are disjoint sets.
- L3 tracks core that last touched cacheline.
 - Read request from a different core cause L3 to retain copy of data in O or S state.
 - i.e. assumes data shared, hence inclusive behavior.
 - Read from same core causes L3 to return data and invalidate cacheline in L3 (not retain a copy).
 - i.e. assumes data not shared, hence exclusive cache behavior.
 - Writes from same core or different cores implemented according to exclusive cache paradigm.



A Few Programming Hints

1. Use SSE2 instructions that modify entire 128bit SSE register instead of preserving one half.
2. Generally good to prefetch 6 to 8 cachelines ahead
 - Latency-Bandwidth product estimates how much data must be “in-flight”
 - 1P, DDR2-800 $\sim 53\text{ns} * 10\text{GB/s} = 530\text{ Bytes} = \sim 8\text{ cache lines in flight}$.
 - 2P, DDR2-667 $\sim 81\text{ns} * 17\text{GB/s} = 1377\text{ Bytes} = \sim 21\text{ cache lines in flight (combined across both Northbridges)}$.
 - 2P, DDR2-800 $\sim 81\text{ns} * 20\text{GB/s} = 1620\text{ Bytes} = \sim 25\text{ cache lines in flight (combined across both Northbridges)}$.
3. Try to have 100 cycles of computation in loop body between successive prefetches
4. Avoid issuing multiple software prefetches to the same cacheline
5. Unroll loops enough times so each iteration works on 1 or more cachelines of data.

note: neither hw or sw prefetches will be allowed to generate page faults, but a TLB miss on a prefetch can initiate a TLB fill.



Programming Hints con't.

Which Prefetch to use ?

Data	Less than $\frac{1}{2}$ L1 size	Less than $\frac{1}{2}$ L2 size or of unknown size		Greater than $\frac{1}{2}$ L2 size
		Reused	Not Reused	
Read only	prefetch or prefetchnta	prefetch	prefetchnta	prefetchnta
Sequential read only	hwprefetcher + prefetch	hwprefetcher + prefetch	prefetchnta	prefetchnta
Read-write	prefetchw	prefetchw	prefetchnta	prefetchnta
Sequential read-write	prefetchw	prefetchw	prefetchnta	prefetchnta
Write only	prefetchw	prefetchw	movnt	movnt
Sequential write only	hwprefetcher + prefetchw	hwprefetcher + prefetchw	movnt	movnt



Performance Case Study 1

SPEC OMPL2001 (SPEC-HPG OpenMP benchmark)
313.swim_I (shallow water ocean model)

Opteron™ (Barcelona) System

Tyan Thunder n425QE (S4985E)
Four Opteron 8356 CPUs @ 2.3GHz
16 x 2GB DDR2-667
SLES10 SP1 X86_64
PathScale Compiler Suite 3.1



Performance Case Study 1 (slide 2)

Three sets of compiler* flags used:

"Ofast" (aka "generally a good set of optimizations")

-mp -Ofast -mcpu=barcelona -OPT:early_mp=on -mcmmodel=medium

"Ofast_simd0" (aka "don't vectorize")

-mp -Ofast -mcpu=barcelona -OPT:early_mp=on **-LNO:simd=0** -mcmmodel=medium

"Ofast_movnti2500" (aka "don't use streaming stores")

-mp -Ofast -mcpu=barcelona -OPT:early_mp=on **-CG:movnti=2500** -mcmmodel=medium

Flags	Runtime in secs. (mins.)
Ofast	7194s (120m)
Ofast_simd0	1736s (29m)
Ofast_movnti2500	1785s (30m)

What's going on? -

Too much of a good thing? (streaming stores or vectorization)



Performance Case Study 1 (slide 3)

Profiling shows

Problem Size:
7701 x 7701 grid, REAL*8
452MB per array (5.8GB total)

Ofast

samples	%	symbol name
598797565	79.1052	__ompdo_calc3_1
90931114	12.0126	__ompreion_calc2_1
48912887	6.4617	__ompreion_calc1_1
17952271	2.3716	__ompdo_MAIN__1
153331	0.0203	__ompdo_calc3z_1

Program Structure:
10 NCYCLE=NCYCLE+1
Calc1 (writes 4 arrays)
...
Calc2 (writes 3 arrays)
...
Calc3 (writes 6 arrays) ←
...
GOTO 10

Thrashing WebUI!

Ofast_simd0

samples	%	symbol name
68233066	34.4688	__ompreion_calc2_1
61859772	31.2492	__ompdo_calc3_1
48931003	24.7181	__ompreion_calc1_1
18617176	9.4047	__ompdo_MAIN__1
132326	0.0668	__ompdo_calc3_2

Ofast_movnti2500

samples	%	symbol name
68314854	34.4881	__ompreion_calc2_1
61749164	31.1735	__ompdo_calc3_1
48932509	24.7031	__ompreion_calc1_1
18770527	9.4761	__ompdo_MAIN__1
132286	0.0668	__ompdo_calc3_2

Oprofile: Counted CPU_CLK_UNHALTED events (Cycles outside of halt state)



Performance Case Study 1 (slide 4)

Performance Counters

Ofast = default

Ofast_simd0 = -LNO:simd=0

Ofast_movnti2500 = -CG:movnti=2500

	default	"-LNO:simd=0"	"-CG:movnti=2500"
CPI	14	2.55	3.7
Clocks(B)	15373	4107	4146
Insts(B)	1100	1618	1096
L3Req	440.000	161.800	142.480
L3Miss	330.000	134.294	134.808
totSSE	792.000	1164.960	789.120
FPadd pipe	583.000	388.320	252.080
absolute(B) FPmult pipe	242.000	210.340	151.248
FPstore pipe	291.500	142.384	113.984
PgOpen	253.000	37.214	36.168
PgClose	233.200	114.878	111.792
PgCflct	114.400	63.102	61.376



Performance Case Study 2

Store-to-Load Forwarding

CodeGen1

```
loop1:
  mov rbx, [rdx]
  add rax, rbx
  mov [rdx], rax
  mov rbx, [rdx+10h]
  add rax, rbx
  mov [rdx+10h], rax
  mov rbx, [rdx+20h]
  add rax, rbx
  mov [rdx+20h], rax
  mov rbx, [rdx+30h]
  add rax, rbx
  mov [rdx+30h], rax
  mov rbx, [rdx+40h]
  add rax, rbx
  mov [rdx+40h], rax
  mov rbx, [rdx+50h]
  add rax, rbx
  mov [rdx+50h], rax
  mov rbx, [rdx+60h]
  add rax, rbx
  mov [rdx+60h], rax
  mov rbx, [rdx+70h]
  add rax, rbx
  mov [rdx+70h], rax
  mov rbx, [rdx+80h]
  add rax, rbx
  mov [rdx+80h], rax
  mov rbx, [rdx+90h]
  add rax, rbx
  mov [rdx+90h], rax
  dec rcx
  jnz loop1
```

```
For (i=0; i < N; i++) {
    Data[i % 10] = Data[i % 10] + cvalue;
}
```

store

load

Store-to-Load forwarding (STLF) feature:

- Load in iteration i may get data forwarded from earlier Store (iteration i-1) if still in Load/Store queue awaiting cache write.
- Without STLF, Store must write cache then load reads cache.

CodeGen2

```
loop1:
  add [rdx], rax
  add [rdx+10h], rax
  add [rdx+20h], rax
  add [rdx+30h], rax
  add [rdx+40h], rax
  add [rdx+50h], rax
  add [rdx+60h], rax
  add [rdx+70h], rax
  add [rdx+80h], rax
  add [rdx+90h], rax
  dec rcx
  jnz loop1
```

Although this testcase is a bit artificial, it has similarities to code that might be found for circular buffers, CRC, etc.

Assembly Format:
Op dst, src



Performance Case Study 2 (slide 2)

Store-to-Load Forwarding

CodeGen1

```
mov rbx, [rdx]
add rax, rbx
mov [rdx], rax
```

3 x86 instructions,
3 macro-ops,
3+ micro-ops ,
2 separate LSQ entries

CodeGen2

```
add [rdx], rax
```

1 x86 instruction
1 macro-op,
3+ micro-ops,
1 LSQ entry

(Remember, 1 loop iteration
contains 10 such snippets)

So what's going on ?!
Turns out we stumbled
on a pathological corner
case for the 2nd Gen.
Opteron™ !

	CodeGen1	CodeGen2
2 nd Gen. Opteron™	~12 cycles/ iteration	~ 22 cycles/ iteration
3 rd Gen. Opteron™	~12 cycles/ iteration	~12 cycles/ iteration



Performance Case Study 2 (slide 3)

Store-to-Load Forwarding (STLF)

CodeGen1 (1)

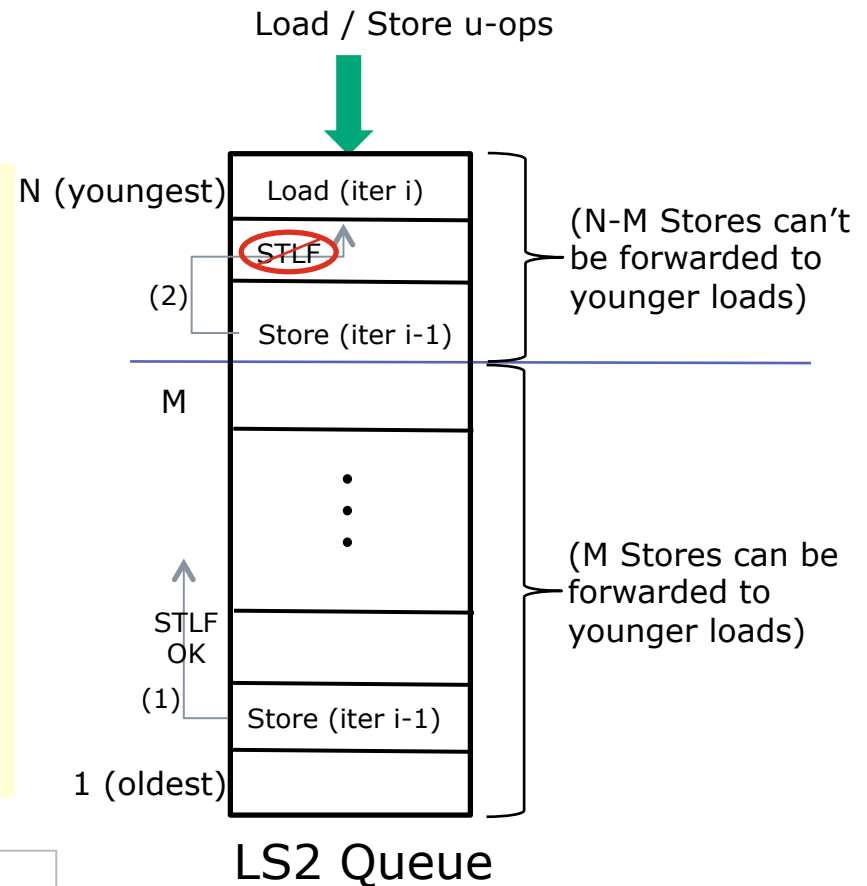
```
mov rbx, [rdx]
add rax, rbx
mov [rdx], rax
```

- Separate load and store causes 2 LSQ entries to be used.
- Fewer loop iterations can accumulate in LSQ due to lower density.
- Lessens chance that load from iteration i and corresponding store from iteration i-1 will both reside in top N-M entries.
- Most STLF opportunities will succeed because store will be in bottom M entries and can forward data.

CodeGen2 (2)

```
add [rdx], rax
```

- Combined ld-op-st instruction means 1 LSQ entry will be used.
- More loop iterations can accumulate in LSQ due to higher density.
- Increases chance that there will be a load op for iteration i that wants store from corresponding instruction in iteration i-1 AND both reside in top N-M entries.
- Since STLF not allowed there, this can cause more pipeline bubbles.




LdSt Queue (LS2) of depth N
STLF supported from bottom M entries ($M < N$)



Performance Case Study 2 (slide 4)

Store-to-Load Forwarding (STLF)

- How would I figure something like this out ?
 - not easily, intuition, unusual performance delta between platforms
 - CrayPAT – unexpected differences in IPC, LS2 full, canceled STLF ops.
- What can I do about it ? 
 - Manually try different unroll factors
 - PGI
 - -Munroll, -Munroll=c:x, -Munroll=n:x, -Munroll=m:x (“x” = unroll factor)
 - Pathscale
 - -CG:load_exe=N (threshold for subsuming loads into arithmetics - produces CodeGen2).
 - -LNO:full_unroll_size, etc.

“Proof of the pudding is in the eating”
Often (intelligently) trying things is
quickest.

Bottom Line: No one size fits all approach, but knowing what’s in your toolbox, allows you to try different things based on intuition, experience.

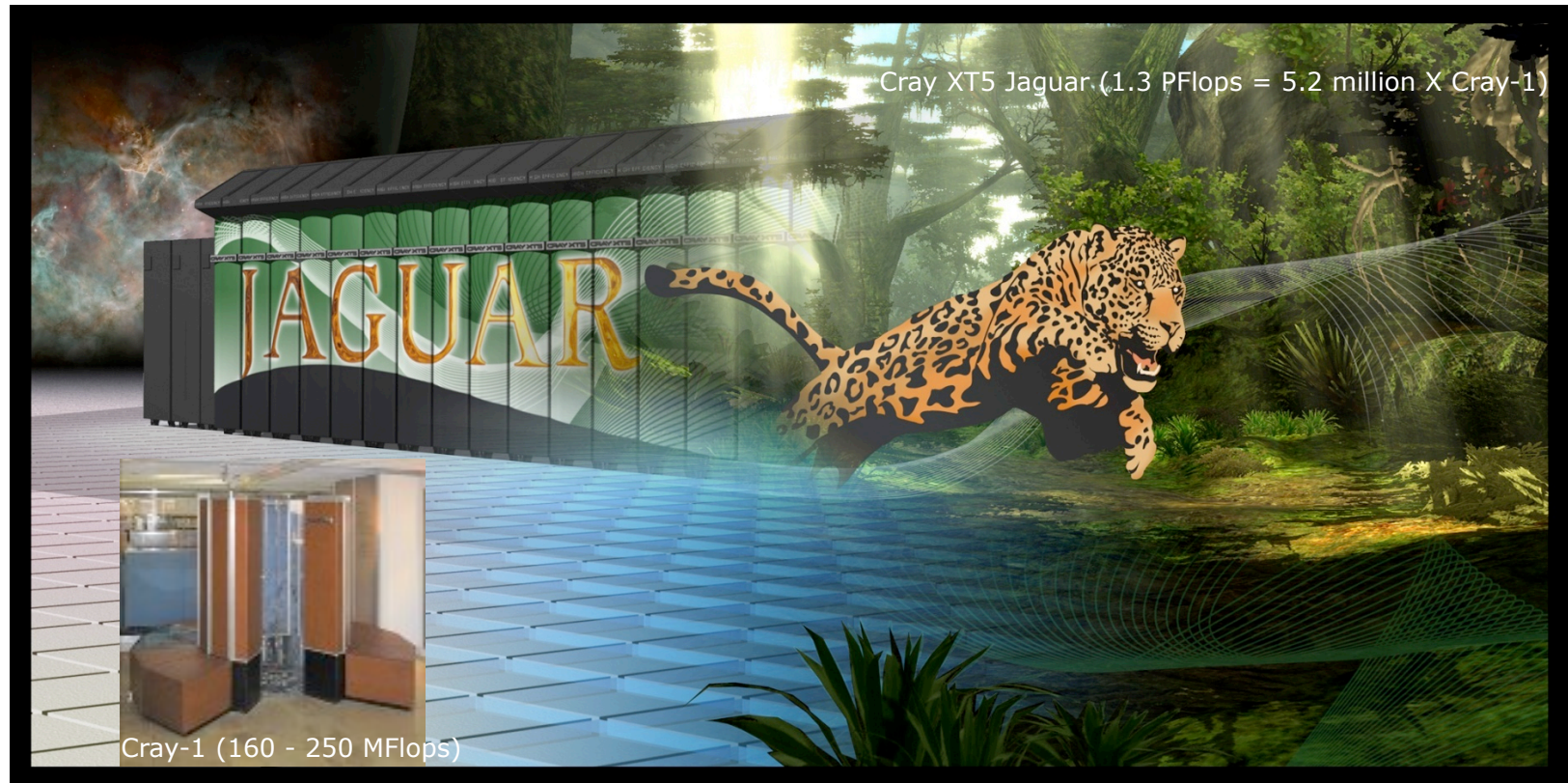


Summary

1. Don't sweat all the details – there is more here than an application writer needs (or should) try to optimize for.
2. High level understanding can help avoid some of the worst case performance pitfalls.
3. Your goal should be to make the compiler and hardware's jobs easier where feasible.
 - Compiler's negative feedback is useful.
 - Performance counters can help.
4. Some Hardware Features to Keep in Mind:
 - Improvements to Core IPC, TLB, HW prefetch, FPU, and memory BW and latency.
 - Flop rich programming environment.
 - Shared L3 (data sharing aware).
 - Be aware of NUMA when coding (have two-socket compute nodes).
 - Many cores and caches requires some awareness regarding cachelines and coherency.



Questions



Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2008 Advanced Micro Devices, Inc. All rights reserved.

